



An adaptive read/write optimized algorithm for Ceph heterogeneous systems via performance prediction and multi-attribute decision making

Zhike Li¹ · Yong Wang^{1,2}

Received: 12 May 2022 / Revised: 12 August 2022 / Accepted: 22 September 2022 / Published online: 5 October 2022
© The Author(s) 2022

Abstract

The advent of the Big Data era has brought considerable challenges to storing and managing massive data. Moreover, distributed storage systems are critical to the pressure and storage capacity costs. The Ceph cloud storage system only selects data storage nodes based on node storage capacity. This node selection method results in load imbalance and limited storage scenarios in heterogeneous storage systems. Therefore, we add node heterogeneity, network state, and node load as performance weights to the CRUSH algorithm and optimize the performance of the Ceph system by improving load balancing. We designed a cloud storage system model based on Software Defined Network (SDN) technology. This system model can avoid the tedious configuration and significant measurement overhead required to obtain network status in traditional network architecture. Then we propose adaptive read and write optimization algorithms based on SDN technology. The Object Storage Device (OSD) is initially classified based on the Node Heterogeneous Resource Classification Strategy. Then the SDN technology is used to obtain network and load conditions in real-time and an OSD performance prediction model is built to obtain weights for performance impact factors. Finally, a mathematical model is proposed for multi-attribute decision making in conjunction with the OSD state and its prediction model. Furthermore, this model is addressed to optimize read and write performance adaptively. Compared with the original Ceph system, TOPSIS_PA improves the performance of reading operations by 36%; TOPSIS_CW and TOPSIS_PACW algorithms improve the elastic read performance by 23 to 60% and 36 to 85%, and the elastic write performance by 180 to 468% and 188 to 611%, respectively.

Keywords SDN · Ceph · Performance prediction · Multi-attribute decision making · Replication scheme

1 Introduction

With the rapid spread of social networking services and the growing number of Internet of Things (IoT) [1] devices, the world's digital information is increasing each year

exponentially. According to market research firm IDC, available data will increase to 180 ZB by 2025 [2]. Many distributed storage systems are being actively developed in industry and academia to efficiently manage to store big data in a scalable and reliable manner. Furthermore, we could develop more practical cloud service discovery mechanisms in cloud environments in the future [3, 4]. Typical storage systems include GFS [5], Ceph [6], Azure Storage [7], Amazon S3 [8], Openstack [9], etc. These storage systems are linked to many common servers to provide storage services to the public. Most of them utilize multiple replicas or corrective codes to achieve high reliability and fast recovery with scalability, high performance, and low cost. Ceph [6] is a reliable, self-balancing, self-recovering distributed storage system that eliminates traditional metadata nodes. It maps data to storage nodes

✉ Zhike Li
lzk18339166123@163.com

Yong Wang
ywang@guet.edu.cn

¹ School of Computer Information and Security, Guilin University of Electronic Technology, Guilin 541004, Guangxi, China

² Collaborative Innovation Center of Education Big Data Application and Network security, Guilin University of Electronic Technology, Guilin 541004, Guangxi, China

through a pseudo-random data mapping function called Controlled Replication Under Scalable Hashing (CRUSH) [10]. Ceph requires only a small amount of local metadata and simple computation to achieve data addressing. The system's scalability has no theoretical limitation, so the industry has widely used it.

The network interconnects the nodes in a large-scale Ceph heterogeneous distributed storage system consisting of thousands of storage nodes. Currently, the network has become one of the bottlenecks limiting performance. However, the developers didn't consider the underlying network conditions, node load, and heterogeneity in the design of Ceph. The data storage location is weighted only by the storage capacity of the nodes [8]. The type of OSD (Object Storage Device), the nodes' computing power, and the nodes' residual bandwidth affect the heterogeneous storage nodes' I/O performance. For example, HDD-type OSD read and write performance is significantly lower than SSD-type. The cluster nodes occupy large network bandwidth when the network load increases (e.g., data migration within the cluster). With limited residual bandwidth and low computing power, the storage node's performance reduces significantly, lowering the overall system's read and write performance. As a result, if we can obtain the network status, load situation, and heterogeneity information of the cluster, we could use it as the weight for selecting OSDs in real-time. The client will dynamically select the best performing OSD or group of OSDs to complete read and write requests. And the performance of the cluster will improve. But the Ceph community did not fully consider how to exploit heterogeneous storage nodes' performance. It is necessary to manually edit the CRUSH Map to adapt to different storage performance requirements [11], which was time-consuming and labor-intensive.

Furthermore, the same CRUSH Map cannot flexibly adapt to situations where the system's read and write performance requirements change dynamically. Therefore, if we can achieve a storage system, it could adaptively provide services for various storage application scenarios. We can reduce the cost of manual editing, achieve optimal allocation of system resources, and improve the quality of service to clients.

Therefore, this paper designs a system architecture that combines the Ceph storage system with Software Defined Network (SDN), based on the traditional Ceph distributed file system. The Ceph cluster runs in an SDN environment. The OSDs are classified using the Node Heterogeneous Resource Classification Strategy. The SDN controller collects information on heterogeneity, network status, and load for each type of OSD. And We use it as the basis for OSD selection in real-time. Following that, we establish the OSD read/write performance prediction model to

dynamically adjust the relationship between the load factor and the performance weight of the OSD. Moreover, we establish a mathematical model for multi-attribute decision-making in conjunction with the OSD load state. Finally, we propose the TOPSIS_PA (Technique for Order Preference by Similarity to Ideal Solution_Primary Affinity), TOPSIS_CW (Technique for Order Preference by Similarity to Ideal Solution_CRUSH Weight), and TOPSIS_PACW (Technique for Order by Similarity to Ideal Solution_Primary Affinity and CRUSH Weight) algorithms to improve the performance of reading/writing operations. TOPSIS series algorithms adaptively optimize the read/write performance of the cluster through a mathematical model. This paper has the following specific contributions:

- (1) Classifying OSDs based on a Node Heterogeneous Resource Partitioning Strategy. We combine the initial node's heterogeneity and network condition as a basis for classifying different types of OSDs;
- (2) Establishing predictive models for the read/write performance of different types of OSDs. We adjust the performance weights of OSDs dynamically with varying states of the load so that we could determine the performance weights with finer granularity;
- (3) Dynamic Pool is proposed to meet a variety of performance requirements. In Dynamic Pool, we establish a mathematical model for multi-attribute decision-making. The TOPSIS_PA, TOPSIS_CW, and TOPSIS_PACW adaptive read/write optimization algorithms are proposed along with the mathematical model.

The structure of this paper is as follows: Chapter 1 is the introduction. Chapter 2 focuses on related work. Chapter 3 analyzes the limitations of CRUSH and heterogeneous resource management in Ceph. Chapter 4 focuses on the architecture and implementation of the system. Chapter 5 conducts the experimental evaluation. Chapter 6 is the conclusion and outlook.

2 Related work

The widely used algorithms are the consistent HASH algorithm [12], the elastic HASH algorithm, and the CRUSH algorithm [10]. Chen Tao et al. [13] of the National University of Defense Technology proposed the CCHDP algorithm. It combines the clustering algorithm with the consistent HASH method to reduce the storage space while introducing a small number of virtual devices and reducing the time consumed for locating data. Jia C J et al. [14] described an essentially perfect hashing algorithm for calculating the position of an element in an

ordered list, appropriate for the construction and manipulation of many-body Hamiltonian, sparse matrices. However, these node selection algorithms do not consider the network load condition when considering the data location mapping relationship. In the open source Ceph cloud storage system, this paper combines the factors of node heterogeneity, node network conditions, and node load in the CRUSH algorithm. Then, to improve the performance of distributed storage systems, we design adaptive read/write performance optimization algorithms for specific storage scenarios.

In dealing with reading/writing load and adaptive performance tuning of Ceph distributed storage systems, Jeong et al. [15] proposed lock contention-aware messenger (Async-LCAM) to dynamically add or remove allocated threads to balance the workload between worker threads by periodically tracking lock contention per connection. Qian et al. [16] proposed a stable accelerated merging-in-memory (MIM) architecture to balance the fluctuation phenomenon of the log file system. Moreover, they improve the system's I/O performance by leveraging the efficient data structure of hash table-based multi-linked tables in memory. Yang et al. [17] used a scaling-based file distribution mechanism to integrate Ceph, HDFS, and Swift storage environments. In this mechanism, subfiles are distributed to different storage services to improve the I/O performance of the system. Kong et al. [18] developed a Mean Time To Recovery (MTTR) and performance prediction model. The model presents good quadratic and linear relationship predictions of performance and MTTR when tuning the recovery and balancing parameters of the Ceph cluster. It provides potentially inefficient parameters that lead to performance loss during recovery. Furthermore, it gives more efficient recovery operations and reference metrics to ensure the reliability and quality of service of the Ceph cluster. However, these load balancing strategies and adaptive performance tuning efforts don't incorporate storage network state considerations.

Based on the above analysis, we need to collect the network status information of storage nodes efficiently and in real-time. However, traditional network state measurement methods require cumbersome configuration and significant measurement overhead [19, 20]. The SDN technology is an emerging programmable network model. The network model separates the control and data planes. Its control level is moving towards centrality and unification. The SDN controller senses global network state, static topology, dynamic flow table information, and other information. The SDN switch simply performs the appropriate actions according to the uniform rules issued by the controller. In a distributed storage network, we could use SDN to obtain network state information of Ceph with less hardware configuration and measurement overhead. This

paper uses the flow measurement method of OpenFlow protocol, SDN's mainstream network measurement function. OpenTM [21] constructs a network-wide flow matrix by obtaining the routing information in the OpenFlow controller and reading the active flow table entry statistics on different switches at regular intervals. Moreover, we obtain the network information and load status of the Ceph cluster in conjunction with the Ryu controller. Research on using SDN technology to optimize the load of storage clusters has emerged. Liberato et al. [22] proposed residue-defined networking architecture as a new approach for enabling key features like ultra-reliable and low-latency communication in Micro Datacenters networks. Kafetzis [23] et al. introduced a bridge called SDR between centralized network control inherent in SDN and the distributed nature of MANETs. The SDR offers the add-on features of flexible and fast PHY and MAC layer adaptation. For solid, autonomous, and ultimately better network control implementations span all layers, the SDR is towards realizing and implementing the holy grail of natural cross-layer optimization. Girisankar et al. [24] designed an SDN architecture that integrates optimization algorithms in the optical data network center and performs the data storage scheduling, effectively reducing cluster load. These studies on using SDN to optimize network routing and link aspects of storage systems illustrate the feasibility of adopting SDN technology in distributed storage networks.

3 Background and motivation

In this section, we present the CRUSH algorithm's limitations, load balancing, and on-demand allocation of heterogeneous resources during data storage on Ceph systems, as well as preliminary ideas and methods for addressing these issues.

3.1 Ceph system data stored process

Figure 1 shows the whole process of client data mapping. The reliability and scalability provided by the Ceph distributed file system don't achieve without supporting the underlying component Reliable Auto-nomic Distributed Object Store [25] (RADOS). The RADOS contains many Object Storage Devices (OSDs) and a small number of Monitor nodes responsible for managing the OSDs. It is responsible for smartly managing the underlying Ceph devices. When storing data, client data are cut and numbered according to fixed-size data objects and mapped evenly to each PG. And the PGs are mapped to a group of OSDs by the CRUSH [10] algorithm.

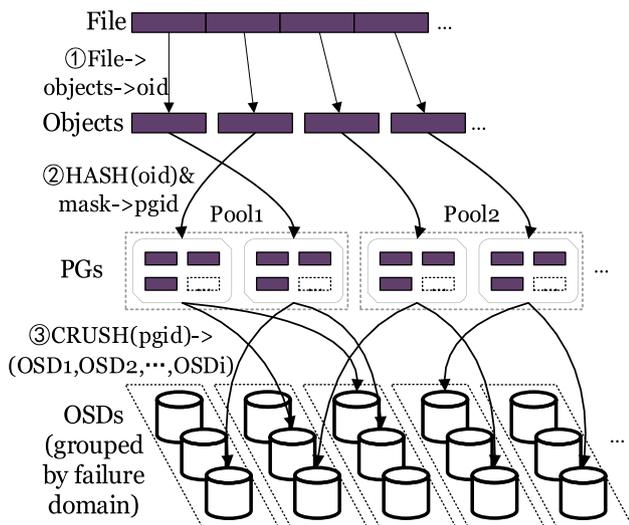


Fig. 1 Ceph cloud storage system data mapping process

The mapping of data objects to PGs and the mapping of PGs to OSDs are the two processes in the whole data mapping path that have the most impact on the data selection OSD in the system. The process of mapping data objects to PGs is:

$$HASH(oid) \& mask = pgid \tag{1}$$

Equation (1) indicates that the object name identifier (oid) obtain by dividing the file. The hash function takes oid as input to generate a random value. The hashed value and the “mask” value are bitwise processed to get the PG number pg_id. In the event of a large number of objects and PGs, this mapping process assures that the distribution of data items is approximately uniform and that the selection is random. Pgid is a structural variable containing pool_id and pg_id, the actual PG number. In the CRUSH algorithm, the mapping process of PG to OSD is:

$$CRUSH(pgid, CRUSH_Map, ruleno) = (OSD_0, OSD_1, \dots, OSD_i) \tag{2}$$

OSD_i denotes the ith object storage device. The number of replicas specified determines the size of i. Pgid denotes the second mapping’s output value. CRUSH Map denotes a cluster map containing information such as the cluster topology. And ruleno denotes the replica placement rule’s number (PPR). The CRUSH algorithm’s inputs include a globally unique PG identity, cluster topology, and CRUSH rules. These inputs control the mapping process to assist PGs in migrating flexibly across multiple OSDs, which enables advanced features such as data dependability and automated balancing.

CRUSH is a hash-based pseudo-random data distribution algorithm developed by improving the RUSH [26]

series algorithm. It enhances efficiency and flexibility while solving reliability and data replication problems by mapping data objects to storage devices in a controlled manner. It also achieves decentralization and avoids the bottleneck effect caused by central nodes in storage systems. The CRUSH algorithm selects storage locations in a pseudo-random way, although it has the following issues and limitations: The input parameters of the storage node selection function `crush_do_rule` include weight vector, replica placement rules, CRUSH Maps, and so on, as can be seen from the CRUSH algorithm’s function call process. The CRUSH Maps record the cluster’s storage hierarchical structure, replica mapping, and proportional weights calculated only from storage capacity. In deciding the PG distribution to the OSD, the weight value is the only limitation on the selection algorithm in different types of Buckets. This storage capacity as the only factor in the decision can only satisfy the uniformity of data distribution in the cluster but ignores the impact of node heterogeneity, including CPU and memory size, different OSD types, underlying network, and node load on the cluster performance. If the heterogeneous cluster has inferior network performance and a high load, selecting overloaded OSDs for the client will degrade the overall cluster performance.

Based on the above analysis, the CRUSH algorithm in Ceph calculates the data distribution based on the storage capacity as the only determinant to obtain the OSD weights. This mapping method can only satisfy the uniformity of spatial data distribution in the cluster but ignores the impact of the underlying network, and OSD load on the cluster read/write performance. It is necessary to optimize the read/write performance of the Ceph system by considering the node’s network state information and the load information. In short, we need to establish an adaptive OSD selection strategy to improve the read/write performance.

3.2 Storage pool with PG

As shown in Fig. 1, the PG is a collection of some data objects. At the architectural level, the PG is the bridge between the client and the ObjectStore. It is responsible for converting all requests from the client into transactions. The ObjectStore can understand these transactions and distribute and synchronize them between OSDs. The ObjectStore completes the actual data storage of the OSD, encapsulating all I/O operations to the underlying storage. Storage pools use PGs as basic units and manage them. To achieve an on-demand allocation of storage resources, RADOS manages the pooling of OSDs in the cluster. A storage pool is a virtual concept that represents a set of constraints. We can design a set of CRUSH rules for a specific storage pool. These rules could limit the conditions

such as the range of OSDs it uses, replica strategies, and fault domains (physically isolated, distinct fault domains).

3.3 Load imbalance and heterogeneous resource allocation issues

In the multi-replica mode of the Ceph storage system, Fig. 2 shows its read operation and write operation, respectively. To ensure data security, Ceph adopts a model of strict consistency writing of objects. When a client requests to write an object, the primary OSD first writes the data object in it. Then the primary OSD sends the data object to each subordinate OSD. The primary OSD does not send the feedback of successful writes to the client until it receives the feedback of successful writes from all subordinate OSD nodes. The primary OSD also writes successfully, thus ensuring the consistency of writing to all replica data. This strict consistency writing strategy leads to long write latency [27]. In addition, when the Ceph cloud storage system performs read operations, only the primary OSD performs read operations. The subordinate OSD nodes are not selected to perform read operations, resulting in relatively high I/O pressure on the master OSD. Moreover, the system can not achieve the read performance of the subordinate OSDs, thus affecting the overall read performance of the cluster.

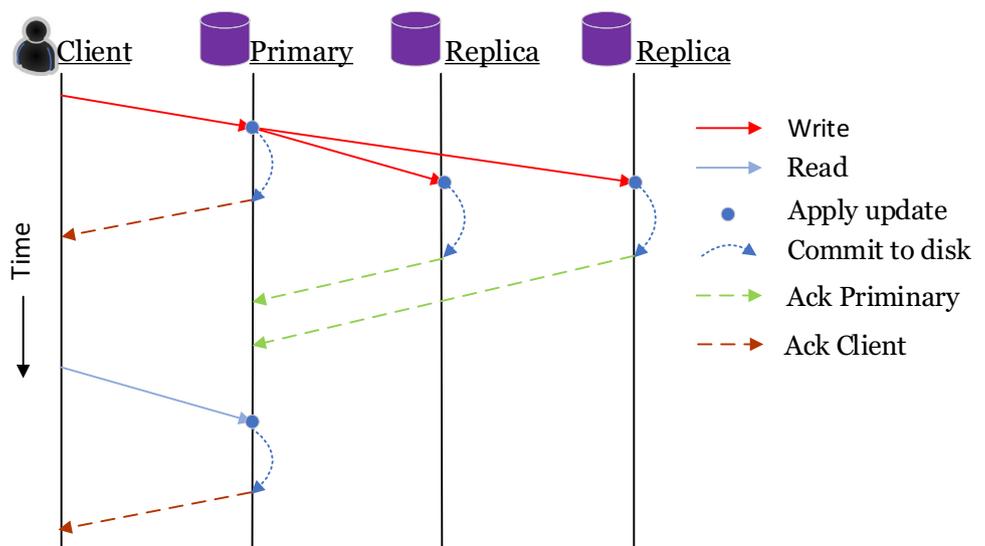
In an extensive Ceph cloud storage system, each OSD must handle I/O operations from multiple clients at the same time. The read and write problems demonstrated in Fig. 2 will magnify and worsen the Ceph cloud storage system's imbalanced load of I/O requests. As a result, a portion of the OSDs will be overloaded or even die,

resulting in a dramatic decline in cluster performance, compromising the cluster's overall quality of service. At the same time, different business scenarios need different read and write performance requirements. However, the classic Ceph cloud storage system does not give a reasonable adaptive strategy to allocate its heterogeneous resources. The heterogeneous resources refer to the OSDs in heterogeneous nodes. These OSDs can provide different performance requirements such as IOPS, throughput, and latency.

Based on the above analysis, Ceph's strict consistency writing strategy restricts client writes to the slowest replica that completes the write operation. The primary replica read strategy focus on client read operations on the primary OSD. The read and write mechanism allows us to optimize the read and write performance of the Ceph cluster in two ways. In storage pools with the same classes OSDs, we dynamically tune read and write requests to concentrate on lower-loaded OSDs. In storage pools with different OSDs, we focus dynamically on client read and write requests on higher-performing OSDs or lower-loaded OSDs.

Furthermore, the heterogeneous resources of Ceph cloud storage systems lack comprehensive management and planning. It requires manual rewriting of the CRUSH Map to adapt to changing storage performance requirements, limiting the Ceph cloud storage system's application scenarios. Suppose the storage system can automatically meet various storage performance scenarios. In that case, it can reduce the cost of manual editing and achieve the best allocation of system resources. The application demand scenarios include high performance read, high performance write, low performance read and write, and other scenarios. If the clients that only need low read and write performance

Fig. 2 Read/write operation process in multi-replica mode



use high-performance read and write OSDs, it will waste system resources.

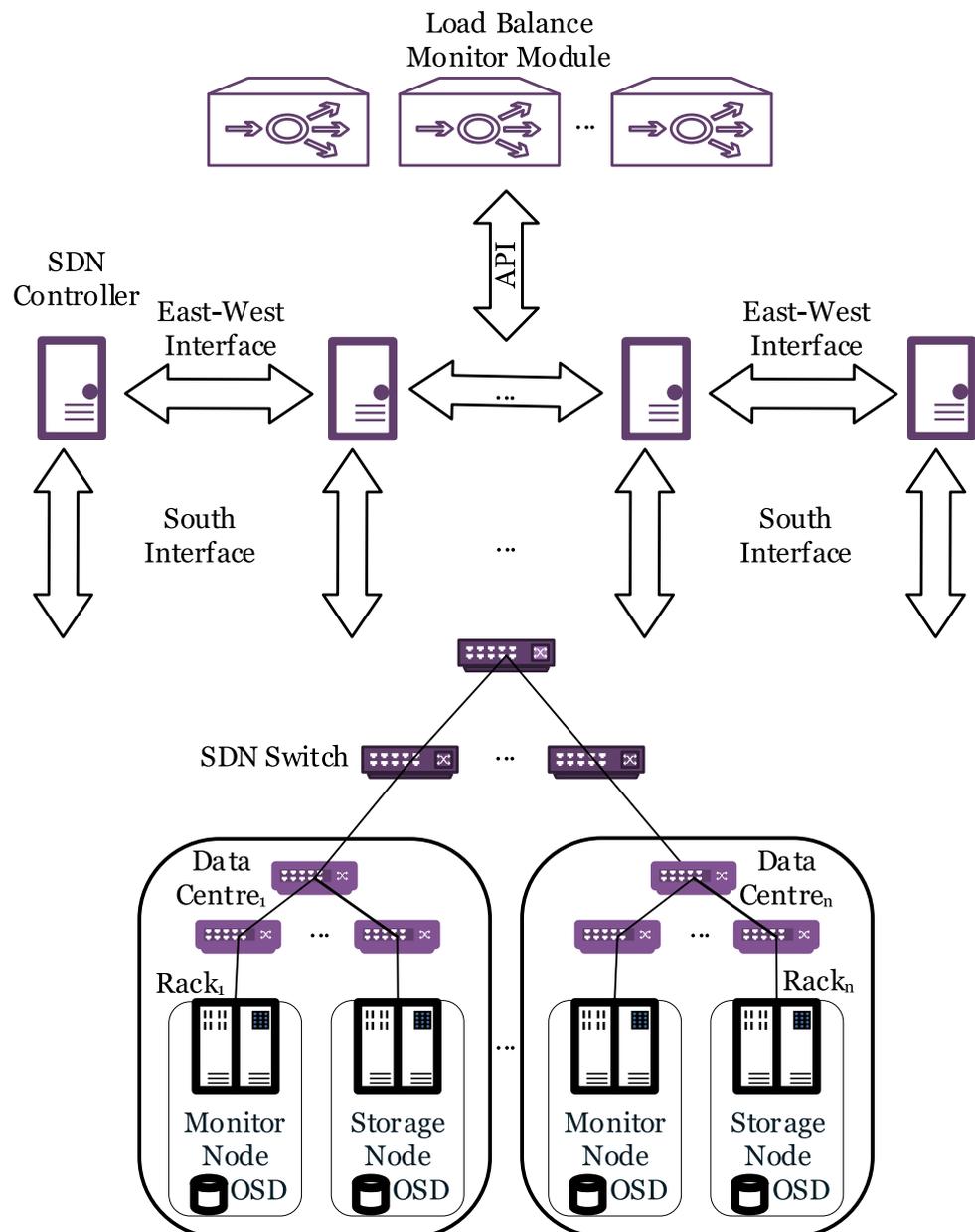
4 System design and implementation

In this section, we present an adaptive read and write optimization model for Ceph heterogeneous storage systems. We first incorporate the architecture of SDN technology. Then, we combine the OSD host node's network state and load information. Finally, we optimize the read/write performance of the cluster from three aspects: the limitations of the CRUSH algorithm, the load imbalance, and storage service scenarios.

4.1 System architecture

Figure 3 shows the architecture of the distributed storage system based on SDN technology. The bottom layer of the system architecture consists of a monitor node and a storage node. Each storage node can contain multiple OSDs, and the monitor node maintains global configuration information for all nodes in the cluster. The OpenFlow switch connects all servers. Moreover, it is responsible for the transfer of data between them. At the top of the system architecture is the Load Balance Monitor Module (LBMM), which monitors the required OSD information using the SDN controller. The monitor node remotely calls the information collected by the SDN controller to build an

Fig. 3 Distributed storage system architecture incorporating SDN technology



OSD performance prediction model to decide to select the OSD on the storage node.

The system architecture utilizes a network model that separates the control plane and data plane to facilitate our monitoring of the underlying network and load conditions. Wang et al. [28] described in detail the design ideas and implementation methods of the network monitoring module based on SDN technology. We can use the SDN controller's active method to get the network's remaining bandwidth. Then we integrate the acquired OSD load state information and bandwidth information. Finally, we send it down to the Monitor node of the Ceph system through the active way of the SDN controller.

4.2 OSD performance impact factors

In this section, we determine the read/write performance weights of OSDs by considering the node heterogeneity, network state, and node load to improve the load balancing of the Ceph system. To accurately reflect the performance of the OSD under node heterogeneity about its load state, the performance metrics selected in this paper are as follows:

- (1) Bandwidth B , number of CPUs C , and memory size M of the nodes. OSDs are intelligent, semi-autonomous devices that consume CPU, memory, and network bandwidth resources to perform fault recovery and automatic data balancing.
- (2) I/O load of OSD L . The I/O load of the OSD disk reflects the real-time read and write situation of this OSD. If the read or write load is high, the delay of OSD in responding to the client's read and write data request will increase, leading to the death of the OSD node in extreme cases.
- (3) The number of OSDs on heterogeneous nodes is H , and the OSD type is T . The literature [29] presented a significant performance difference between SSD and HDD type OSDs in handling client read and write operations.
- (4) The number of PGs occupied by the OSD is P . Two implementations of the Object-Store, FileStore, and BlueStore, provide APIs for reading and writing threads, both at a PG granularity. Moreover, the proportion of PGs occupied by the OSD directly affects the proportion of the read and write load allocated to the client.

We choose these metrics to consider a combination of OSD network state and load conditions. These metrics are not necessarily correlated, especially in heterogeneous networks and nodes. For example, with different bandwidths and node processing capabilities, the residual bandwidth of each heterogeneous OSD node will not necessarily be related to the I/O load of the node. The remaining bandwidth is related to the current service volume and hardware infrastructure. Even if the same host load does not tell us residual bandwidth in the network, this situation has also been discussed in the literature [30]. It demonstrates that memory and CPU utilization correlate in some cases, but I/O load and CPU are not in some cases. Therefore, the above selected OSD impact factors are not necessarily related, and the selected dimensions provide a better picture of the network condition and load of the OSD nodes.

4.3 Node heterogeneous resource division strategy

Algorithm 1: OSD heterogeneous resource partitioning algorithm

Input: OSD information (OSD_Info)

Output: OSD Minimal Classification Set (χ)

```

1:  $OSD\_Info = get\_osd\_info()$ 
2: Procedure GetOSDPerfFactor( $OSD\_Info$ )
3:    $\alpha = \{\}$ 
4:   for  $osd$  in  $OSD\_Info$  do
5:     for  $e_i$  in  $OSD\_Info[osd]$  do
6:       if  $e_i$  is 'expect' then
7:          $\alpha[osd][e_i] = OSD\_Info[osd][e_i]$ 
8:   end procedure
9: procedure GetOSDMiniClassifiSet( $\alpha$ )
10:   $\beta = \{\alpha_i\}, \chi = \{\}$ 
11:  for  $osd$  in  $\alpha$  do
12:    if  $\beta \cup \alpha[osd] \neq \beta$  then
13:       $\beta = \beta \cup \alpha[osd]$ 
14:       $\chi = \chi \cup osd$ 
15:  end procedure

```

Algorithm 1 shows the main steps of the Node Heterogeneous Resource Partitioning Strategy. Line 1 is to get the heterogeneous OSD information of the nodes in the Ceph

system. In line 2, select i ($1 \leq i \leq 7$) performance metrics according to the performance requirements. Then traverse the initial heterogeneous performance set $\alpha_j = \{e_1, e_2, \dots, e_i\}$, and generate $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_j\}$, $1 \leq i \leq t$, where e_i denotes the initial value of the j th OSD performance metric, t is the total number of OSDs in the Ceph. In line 9, initialize the OSD minimal performance set $\beta = \{\alpha_1\}$ and the OSD minimal classification set $\chi = \{\}$, and continue to traverse the set α . If $\beta \cup \alpha_j \neq \beta$, then $\beta = \beta \cup \alpha_j$, and $\chi = \chi \cup \text{osd}$; otherwise β remains unchanged. Finally, Algorithm 1 generates the OSD performance set $\beta = \{\alpha_1, \alpha_2, \dots, \alpha_l\}$ and the OSD classification set $\chi = \{\text{osd}_1, \text{osd}_2, \dots, \text{osd}_l\}$, where $1 \leq l \leq t$, osd_l is the number of OSDs corresponding to α_l .

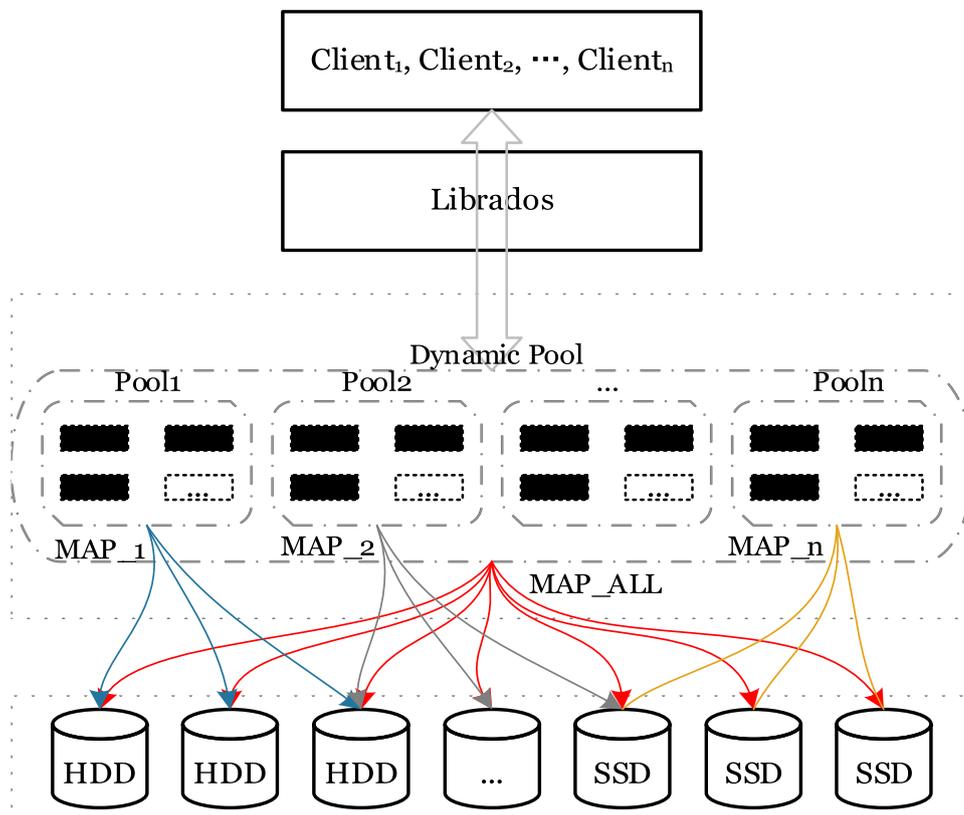
As shown in Fig. 4, we divide the Ceph heterogeneous storage system to obtain different performance storage pools. $\{\text{Pool}_1, \text{Pool}_2, \dots, \text{Pool}_n\}$ logically bundles the OSDs into separate storage pools with gradually increasing performance. Pool_1 is the Pool with the worst read/write performance. Moreover, it is generally composed entirely of HDD-type OSDs with poor network performance and few node resources.

Pool_2 is between Pool_1 and Pool_n in terms of reading/writing performance. And it may be composed of HDD-Type and SSD-Type OSDs, with HDD-type OSDs located on nodes with many resources and good network performance and SSD-type OSDs situated in nodes with the opposite. Pool_n is the opposite of Pool_1 in terms of OSD performance. Furthermore, Pool_n is generally made up entirely of SSD-type OSDs. And it includes a good network and nodes with more resources. Dynamic Pool can provide dynamic read/write performance without changing the CRUSH Map. It provides clients with different read and write performance requirements and achieves the best allocation of system resources.

4.4 OSD load monitoring strategy

Algorithm 2 is responsible for obtaining the OSD load status and runs on the cluster’s OSD nodes. It passively receives messages forwarded from the switch via the SDN controller. It parses the packets in the messages to collect the OSD status.

Fig. 4 Divided performance storage pool



In line 1, the input parameter to the GetOSD() function is the host IP of all nodes. We obtain the OSD of each node and form a dictionary OSD {osd: host_ip} to record the mapping relationships. In line 4, the GetOSDInfo() function collects the node's CPU usage, memory usage size, and the number of OSDs. Moreover, it collects the type, the number of occupied PGs, and the I/O load of the OSD and records it in the dictionary OSD_Load_Info{}. In line 14, the SendData() function sends this load data to the switch via a UDP message, with the destination ADDR being the user-specified host IP.

Based on Algorithm 2, the SDN controller delivers the UDP packets containing the OSD load information via Packet-In messages. The Ryu controller running in the SDN network maintains the dictionary ip_to_port {(dpid, in_port), src_ip}, which records the IPs of hosts connected under different switches ports. When the Ryu controller receives a Packet-In message [28], it first determines whether the Ceph Monitor node sends the message based on the dictionary ip_to_port. If so, it records the packet's msg.datapath information as the path to the Packet Out message. Then the IPv4 and UDP packet header protocols are parsed by get_protocols() to determine if they are the specified ADDR and port number.

Algorithm 2: OSD load collection algorithm

Input: All host IP information in Ceph (*Host_IP*)

Output: OSD load information (*OSD_Load_Info*)

```

1: Procedure GetOSD(Host_IP)
2:   OSD ← Host_IP
3: end procedure
4: procedure GetOSDInfo(OSD)
5:   OSD_Load_Info = {}
6:   for i in OSD do
7:     OSD_Load_Info[i] ← cpu[Host_IP[i]]
8:     OSD_Load_Info[i] ← mem[Host_IP[i]]
9:     OSD_Load_Info[i] ← host[Host_IP[i]]
10:    OSD_Load_Info[i] ← type[Host_IP[i]]
11:    OSD_Load_Info[i] ← pgs[Host_IP[i]]
12:    OSD_Load_Info[i] ← io[Host_IP[i]]
13: end procedure
14: procedure SendData(OSD_Load_Info)
15:   send(IP(src = 'src_ip', dst = 'dst_ip') /
        UDP(dst_port) / Raw(OSD_load_Info))
16: end procedure

```

Algorithm 3: Sending OSD load information via Packet-Out packets

Input: OSD load information received via Packet-In packets

Output: Packet-Out packets with OSD load information

```

1: procedure: send_packet_out(msg, Data)
2:   datapath ← msg.datapath
3:   ofproto ← datapath.ofproto
4:   ofp_parser ← datapath.ofproto_parser
5:   pkt ← packet.Packet(msg.data)
6:   eth_header ← pkt.get_protocols
   (ethernet.ethernet)[0]
7:   dst_mac ← eth_header.src
8:   arp_header = pkt.get_protocols(arp.arp)
9:   dst_ip = arp.header.src_ip
10:  out_port = msg.match[in_port]
11:  ether_instance = ethernet.ethernet(dst_mac,
   src = controller_mac, eth_header.ethertype)
12:  ipv4_instance = ipv4.ipv4
   (src = controller_ip, dst = dst_ip, proto = 17)
13:  udp_instance = udp.udp(src_port = 12345,
   dst_port = 10086)
14:  pkt = packet.Packet()
15:  pkt.add_protocol(ether_instance)
16:  pkt.add_protocol(ipv4_instance)
17:  pkt.add_protocol(udp_instance)
18:  pkt.add_protocol(data)
19:  actions = [ofp_parser.OFPActionOutput
   (out_port)]
20:  req = ofp_parser.OFPPacketOut(datapath,
   buffer_id = ofproto.OFP_NO_BUFFER, actions,
   in_port = ofproto.OFPP_CONTROLLER, data)
21:  datapath.send_msg(req)
22: end procedure

```

After a successful match, the `_parse_udp(dpid, port, msg_data)` function [28] decodes the packet content. It identifies the host sending the message based on the input port and dpid of the message. Since the OSD {osd: host_id} records the mapping between hosts and OSDs, we transcribe the bandwidth and load information of hosts into the bandwidth and load information of each OSD.

Algorithm 3 is the process by which the Ryu controller packages the OSD load information and sends it to the Ceph monitor node via Packet-Out packets. In Line 1, the

Ryu controller integrates the data containing OSD load and bandwidth information into a Data dictionary. Then calls `add_protocol()` function to construct UDP packets. Finally, we send out `Packet_Out` messages based on dictionary `ip_to_port {}`.

4.5 Adaptive read/write optimization algorithm based on performance prediction and TOPSIS model

4.5.1 OSD read/write performance prediction model

In this subsection, we consider the complex non-linear relationship between OSD performance metrics and reads/writes performance. Then we select a random forest as the model for building OSD performance prediction and dynamically obtaining the impact of different load factors on OSD performance. Random forest is a powerful integrated model and an extension of the bagging algorithm. It combines the advantages of statistical inference and machine learning methods [31] to make predictions based on a set of regression or classification trees rather than a single tree. Moreover, it combines the outputs of each tree to obtain the final output, which makes the performance prediction accurate and builds a more stable model. The literature [32] compares the error rates of Random Forest (RF), Support Vector Machine (SVM), Artificial Neural Network (ANN), and K-Nearest Neighbors (KNN) predictions. The random forest algorithm builds models that perform significantly better than other machine learning algorithms. In addition, the method is robust to over-fitting and makes no assumptions about the predictor variables.

Figure 5 illustrates the OSD read/write performance prediction model. It consists of three main phases:

Step 1. Heterogeneous resource classification stage. Based on the node Heterogeneous Resource Partitioning Strategy, we obtain the OSD minimal classification set $\chi = \{osd_1, osd_2, \dots, osd_l\}$, where $1 \leq l \leq t$, osd_l is the number of OSD.

Step 2. OSD load information collection stage.

- (1) The Crush Weight (CW) value of all OSDs in the cluster is set to 1 to initialize the uniform distribution of PG to all OSDs so that the read and write performance loads converge. Reset counter n , where n takes on the value range $[1, l]$;
- (2) Take the step of CW as s and set the acquisition time interval as t . For each OSD in the set χ , the CW value gradually increases by s until the IOPS of the cluster no longer grows, or the expected performance requirements are met. Then use the active way of SDN controller to achieve the periodic acquisition of network usage bandwidth of Ceph cluster nodes.

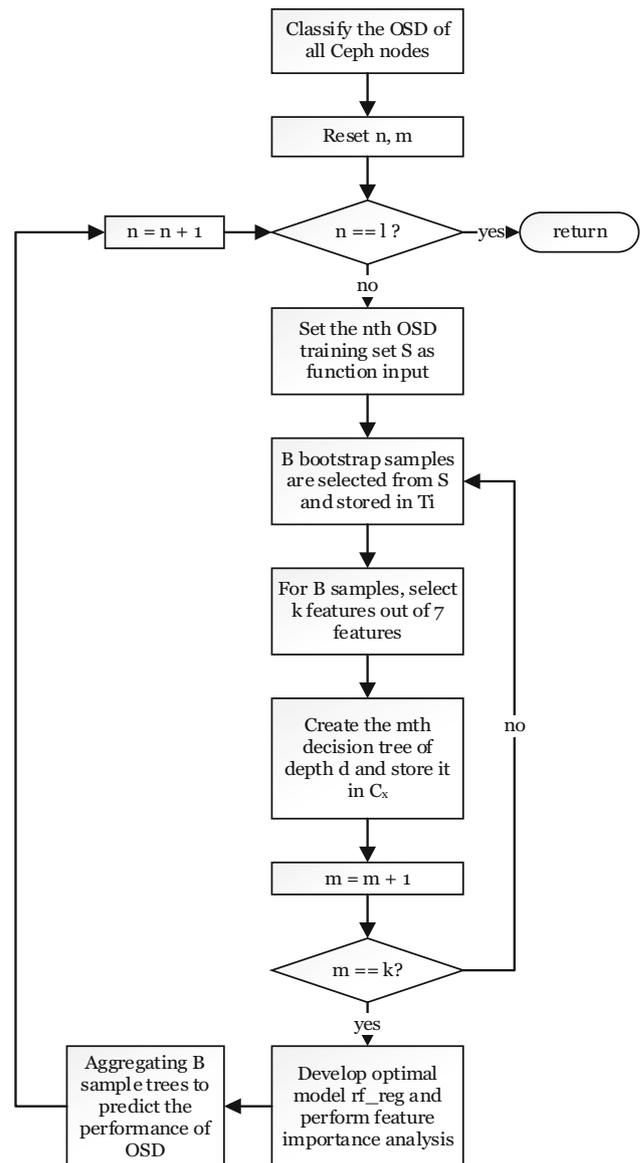


Fig. 5 OSD Performance Prediction Model

Firstly, we use the passive way to obtain the load state information of OSDs, including the node's used bandwidth, the CPU usage, the memory usage size, the number of PGs occupied by the OSD, and the amount of real-time I/O data (in kB) of the OSD. Furthermore, the algorithm begins to generate the parameter sets $consume_i = \{bw_{i1}, cpu_{i2}, mem_{i3}, pgs_{i4}, w_{io_{i5}}\}$ and r_{io_i} or $consume_i = \{bw_{i1}, cpu_{i2}, mem_{i3}, pgs_{i4}, r_{io_{i5}}\}$ and w_{io_i} . Moreover, we obtain the current read and write performance of the cluster $IOPS_i$. The $consume_i$ is a set of resource consumption sets. The r_{io_i} and the w_{io_i} are the corresponding OSD read and write performance. Secondly, we construct the vector set $S =$

$\{\{consume_1, r_{io_1}, IOPS_1\}; \dots; \{consume_p, r_{io_p}, IOPS_p\}\}$ or $S = \{\{consume_1, w_{io_1}, IOPS_1\}; \dots; \{consume_p, w_{io_p}, IOPS_p\}\}$ of the n th OSD. The parameter p is the number of elements in the vector set S of each OSD generated;

- (3) The SDN controller sends down the load information in the vector set S to the Monitor node of Ceph through Packet-Out. Finally, the Monitor node uses Random Forest to build the performance prediction model of the OSD.

Step 3. Building the OSD performance prediction model stage.

- (1) Using the vector S as the input to the random forest, bootstrap samples of size B are selected from the entire sample and stored in T_i ;
- (2) The number of sample features is set to 5, and k features out of the five feature numbers are selected for the B bootstrap samples. And the best segmentation points are obtained by building a decision tree, which is repeated k times to generate k decision trees and store them in C_x ; the maximum depth of the decision tree is set to d ;
- (3) The model `rf_reg` corresponding to the parameters with the best effect is formulated. And the feature importance analysis of OSD performance indicators is performed to obtain the corresponding feature weights of OSDs. Then we aggregate the predictions of B bootstrap sample trees to predict the new performance of OSD pre_{io_i} . pre_{io_i} is r_{io_i} or w_{io_i} .

In this subsection, we build an OSD performance prediction model based on the load monitoring strategy of SDN technology and random forest. It can better understand the relationship between OSD’s read and write performance and consumption. For example, when the Ceph system processes I/O requests from clients, we could use the nodes’ remaining resources to optimize read/write performance accordingly. This model also prepares the next section for building a multi-attribute decision model.

4.5.2 Multi-attribute decision model based on TOPSIS

When the prediction model `rf_reg` accuracy reaches the desired value, we can obtain the OSD’s feature weights by feature importance analysis. This model can reflect accurately the impact of the network state and load factor on its performance.

We obtain the integrated performance weights of the OSDs based on the feature weights. Then we select the best OSD or set of OSDs. This problem can attribute to the multi-attribute decision problem in mathematical

coordination. The Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) [33] is an effective multi-attribute decision-making scheme. The main calculation process is first to normalize the indicators, assign weights according to their importance and construct a weighted normalization matrix. Then, the ideal solution is determined. And the maximum and minimum values of the parameters of each indicator are selected from the weighted normalization matrix. Finally, we calculate the distance between the positive and negative ideal solutions to find the relative closeness of each solution. In the Ceph system, we first classify the OSDs. Then we build a TOPSIS model to optimize the performance of the different storage pools. For each type of resource pool, except for storage capacity, other indicators to measure the merits of OSDs in the remaining bandwidth B , CPU remaining size C , memory remaining size M , and PG ratio P are positive indicators. The larger the value, the better the OSD’s network performance and processing I/O capability. I/O load L is a negative indicator. The smaller the value, the better the performance of the OSD. The negative sign in building the TOPSIS model indicates negative indicators. The designed OSD selection algorithm is as follows:

Step 1. We construct the OSD weight factor decision matrix X in Eq. (3). Then we obtain the normalized decision matrix X' by normalizing Eq. (4). The f_{ij} is the element in the matrix X , i and j are the row and column numbers, respectively, and n is the number of OSDs.

$$X = \begin{bmatrix} B_1 & C_1 & M_1 & P_1 & -L_1 \\ B_2 & C_2 & M_2 & P_2 & -L_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ B_n & C_n & M_n & P_n & -L_n \end{bmatrix} \tag{3}$$

$$X'_{ij} = \frac{f_{ij}}{\sqrt{\sum_{i=1}^n f_{ij}^2}}, i = 1, 2, \dots, n; j = 1, 2, \dots, 5 \tag{4}$$

Step 2. We obtain weights for performance impact factors based on the OSD read and write performance prediction model. The remaining bandwidth B , the occupied PG ratio P , the I/O load L , the CPU utilization C , and the memory utilization M affect OSD performance differently. The relative weight of the remaining bandwidth, the OSDs occupied PG ratio, and the I/O load is significant.

The weighting coefficients W is:

$$W = [W_B W_C W_M W_P W_L] \tag{5}$$

The normalized weighted decision matrix Z is:

$$Z_j = W_j \times X'_{ij}, i = 1, 2, \dots, n; j = 1, 2, \dots, 5 \tag{6}$$

Step 3. The positive and negative ideal solutions of the weighted decision matrix Z are:

$$Z^+ = (Z_1^+, Z_2^+, \dots, Z_5^+) = \max\{Z_{ij} | j = 1, 2, \dots, 5\} \tag{7}$$

$$Z^- = (Z_1^-, Z_2^-, \dots, Z_5^-) = \min\{Z_{ij} | j = 1, 2, \dots, 5\} \tag{8}$$

Step 4. The distances D^+ and D^- for each OSD to the positive and negative ideal solutions are:

$$D^+ = (D_1^+, D_2^+, \dots, D_n^+)$$

$$D_i^+ = \sqrt{\sum_{j=1}^5 (Z_{ij} - Z_j^+)^2} \tag{9}$$

$$D^- = (D_1^-, D_2^-, \dots, D_n^-)$$

$$D_i^- = \sqrt{\sum_{j=1}^5 (Z_{ij} - Z_j^-)^2} \tag{10}$$

Step 5. Calculate the relative read or write closeness C_i^+ of each OSD to the optimal OSD pair, with larger values indicating the better performance of the OSD.

$$C_i^+ = \frac{D_i^-}{D_i^+ + D_i^-}, i = 1, 2, \dots, n \tag{11}$$

Building on the original CRUSH algorithm, we consider five metrics for network performance and OSD load. Moreover, we use these constraints as the OSD weighting factors when selecting an OSD for a read or write operation. Then we could acquire the optimal OSD or set of OSDs by building and solving a multi-attribute decision model.

4.5.3 Adaptive read/write optimization model

In the multi-replica application scenario of the Ceph heterogeneous system, we hope to obtain uniform data distribution and efficient read and write performance when processing read requests. We analyze the adaptive read optimization model by two primary and subordinate OSD selection processes. Phase 1: To meet the cluster’s balanced storage space, Ceph selects a set of OSDs for PGs with storage capacity as the weight. Phase 2: After all PGs have selected OSDs, the TOPSIS model is used to calculate the relative read proximity of OSDs. The OSDs are scored for reading performance and stored as a dictionary OSD_Perf_Set {osd: C_i^+ }.

We can adjust the OSD’s PA (Primary Affinity) value with a value of [0, 1]. The PA value determines the probability that the OSD will become the primary OSD. The primary and subordinate OSD selection module runs on the monitoring node of Ceph. We add a customized OSD_INFO_Map to the Cluster Map maintained by Ceph to record the network status and load information of all OSDs in the cluster. In the second stage of primary and subordinate OSD selection, we update the PA value of the OSD according to the information in the OSD_INFO Map.

Furthermore, we propose the TOPSIS_PA (Technique for Order Preference by Similarity to Ideal Solution_Primary Affinity) algorithm.

We analyze and build the adaptive write optimization model from two phases of the complete data write mapping path. Phase 1: Map data object to PGs. After the client data has been split and numbered according to a fixed size, we can obtain data object identifiers. A pseudo-random function uses these identifiers to map the data evenly to the PGs. Phase 2: Map PGs to OSD. The TOPSIS model uses the pgid, CRUSH_Map, ruleno, and OSD_INFO_Map to obtain the OSD relative write proximity C_i^+ to score the write performance of the OSD and store it as a dictionary OSD_Perf_Set{osd: C_i^+ }.

We can adjust the OSD’s CW (Crush Weight) value to determine where the PG completes the distribution on the OSD again. In the second stage of the data write mapping path, we update the CW value of OSD according to the information in OSD_INFO Map. Moreover, we propose the TOPSIS_CW (Technique for Order Preference by Similarity to Ideal Solution_ CRUSH Weight) algorithm. When we update OSD’s CW value, the cluster will cause the bandwidth consumption of client data migration and data recovery. So we divide Ceph’s RADOS cluster network into a public network and a cluster network for deployment.

Algorithm 4 is the TOPSIS series of performance optimization algorithms, including TOPSIS_PA, TOPSIS_CW, and TOPSIS_PACW, to optimize the read, write and read/write performance of the Ceph system, respectively. TOPSIS_PACW indicates that both TOPSIS_PA and TOPSIS_CW are used to optimize read and write performance. In line 1, we call the GetOSDPerf() function to get the relative proximity of each OSD using the TOPSIS model. If an OSD is “down”, it will not participate in this optimization. In line 7, we call the SetOSDPaCw() function to update the OSD’s PA value or CW value. If the Optimize_Pool flag bit is equal to 0, the value of OSD_Perf_Set directly updates to the PA or CW value corresponding to the OSDs. Otherwise, we select the OSDs in the specified resource pool and find the OSD with the largest value of osd_perf_set and add up its PA or CW value in steps s, where $0 \leq s \leq 1$. The program calls the OSD performance prediction model to predict whether the IOPS value of the OSD is optimized. If it is not, the program will cancel the adjustment. The values 0, 1, and 2 of Select indicate the execution of TOPSIS_PA, TOPSIS_CW, and TOPSIS_PACW optimization algorithms, respectively. Type is the mapping dictionary of OSD number and disk type.

This section describes the architecture and implementation of a distributed storage system based on SDN technology. We first classify the OSDs based on Node Heterogeneous Resource Partitioning Strategy. Then we

monitor the underlying network and load status of the Ceph cluster. We design packet-in and Packet-out algorithms of SDN to monitor the OSD's load status. Finally, we establish the OSD's read/write performance prediction model and the multi-attribute decision-making model. And we propose the TOPSIS series of algorithms to improve the read and write performance of Ceph heterogeneous storage systems.

Algorithm 4: TOPSIS series of algorithms

Input: OSD_INFO_Map

Output: OSD_Perf_Set

```

1: procedure GetOSDPerf(OSD_INFO_Map)
2:   OSD_Perf_Set[osd]=
   TOPSIS(OSD_INFO_Map)
3:   for osd in OSD_Perf_Set do
4:     if osd is 'down' then
5:       remove osd from osd_addr
6:   end procedure
7: procedure SetOSDPaCw(OSD_Perf_Set)
8:   Select = get_optimize_algorithm()
9:   if Optimize_Pool is 'o' then
10:    for osd in OSD_Perf_Set do
11:      If Select is 'o' then
12:        osd_primary_affinity=
        OSD_Perf_Set[osd]
13:        update {osd: primary affinity}
14:      elif Select is '1' then
15:        osd_crush_weight=
        OSD_Perf_Set[osd]
16:        update {osd: crush weight}
17:      elif
18:        osd_primary_affinity=
        OSD_Perf_Set[osd]
19:        osd_crush_weight=
        OSD_Perf_Set[osd]
20:        update {osd: primary affinity,
        crush weight }
21:    else
22:      OSD_Type = get_osd_type()
23:      for osd in OSD_Perf_Set do
24:        if Type[osd] is Optimize_Pool then
25:          If Select is 'o' then
26:            osd_primary_affinity += s
27:            update {osd: primary affinity}
28:          elif Select is '1' then
29:            osd_crush_weight += s
30:            update {osd: crush weight}
31:          elif
32:            osd_primary_affinity += s
33:            osd_crush_weight += s
34:            update {osd: primary affinity,
            crush weight}
35:  end procedure

```

5 Experimental evaluation

5.1 Experimental setup

In this section, we demonstrate the impact of the designed prototype on its performance under an actual experimental platform. The source code is available at <https://github.com/ZhikeLi/SDN-Ceph.git>. Moreover, we verify the effectiveness of the adaptive read and write optimization algorithm for Ceph heterogeneous storage systems. The Ceph cluster in the testing environment has six physical machines of the X86 architecture. The cluster realizes the deployment of SDN multi-controllers to support network communication services for an extensive cloud storage system. We divide the system network into a public network and a cluster network. The public network is responsible for communication between the client and the cluster, the cluster network is responsible for communication among OSDs for data recovery and migration. The Ceph cluster has three physical machines: monitoring and storage nodes. The other three physical machines are storage nodes. One, two, and three of them contain one OSD (SSD disk), two OSDs (one SSD disk and one HDD disk), and three OSDs (all HDD disks), respectively. Table 1 lists specific hardware configuration and the software environment: Ceph version number 14.2.15, Ryu version 4.8, OpenFlow 1.3 protocol, and CentOS Linux release 7.4.1708.

To stress test and verify the storage system, a mainstream benchmarking tool, Fio (version 3.7), is used for performance evaluation in this paper. We formulate 4 KB, 16 KB, 64 KB, 256 KB, and 1024 KB data objects for testing in our experiments. And we use four workloads: random write, random read, sequential write, and sequential read. For the FIO benchmark, we set the iodepth parameter to 128 and numjobs parameter to 8 to generate high I/O traffic in the cluster.

In the test, the Storage Pool of the Ceph cluster sets the number of replicas to 2 and the number of PGs to 512, setting the value generally to an integer around $OSDs \times 100 / Replicas$. The parameter OSDs is the number of OSD. The parameter Rpllicas is the number of replicas. The number of PGs must be an integer power of 2.

5.2 Heterogeneous system resource division

The cluster data collected includes 12 OSDs and four types of heterogeneous OSD resources. Firstly, based on the Node Heterogeneous Resource Classification Strategy, Table 2 shows the results of the OSD heterogeneous resource division. OSD_Type indicates the category number of the classified OSDs. Five performance metrics are

selected: node bandwidth limit (GB/s), number of CPUs, memory size (G), number of node OSDs, and OSD disk type (1 for HDD, 2 for SSD). By collecting OSD_Info data, Algorithm 1 constructs the OSD minimal performance set $\beta = \{\{1, 8, 8, 3, 1\}, \{1, 16, 8, 1, 2\}, \{1, 16, 8, 2, 1\}, \{1, 16, 8, 2, 2\}\}$ and OSD minimal classification set $\chi = \{6, 9, 10, 30\}$.

5.2.1 Discover the potential of heterogeneous system resources

To demonstrate how the TOPSIS series of algorithms affect the performance of heterogeneous clusters, we pick two from each class of OSDs in Table 2, generating sets $\delta = \{4, 6, 8, 9, 10, 16, 7, 30\}$. We test the impact on the performance of clusters of heterogeneous OSDs in set δ . We then scale to more complex heterogeneous environments. In the original heterogeneous cluster, the initial value of PA of the OSDs ranges from [0,1]. The smaller the PA value, the smaller the read load on the OSDs. The CW value of the OSD is a positive correlation weight converted from storage capacity. The larger the CW value of the OSDs, the more PGs they carry. Thus the more read and write load the OSDs need to carry. Figures 6 and 7a, b, c, and d show the normalized throughput curves for simultaneous 100% random read operations and 100% random write operations for the same RBD block, respectively. The experimental test read/write object size is 16 KB. The horizontal

coordinate 1 of Fig. 6b and Fig. 7b denotes the random read and random write normalized throughput of the original cluster, respectively. We normalize the random read and write throughputs represented by coordinate 1 to 1 as the baseline throughput. We test the impact of OSD_Type, PA value, and CW value on the cluster performance and compare it with the benchmark throughput.

As shown in Fig. 6b, at horizontal coordinate 2 we update the PA value of the first class of OSDs to 0.1. This operation means that we reduce the read load on the first class of OSDs while concentrating more read load on the other three classes of OSDs with better read performance. The read performance improves by 5%. At horizontal coordinate 3, we update the PA of type 1 and type 2 OSDs to 0.1. Read performance improves by 20%. In horizontal coordinate 4, we update the PA values of type 1, type 2, and type 3 OSDs to 0.1 so that the type 4 OSDs concentrate more on the read load. Read performance improves by 12%. Overall, the cluster read performance increased and then decreased as I kept updating the PA values for the different categories of OSDs. This operation is because, with a constant write load distribution, we concentrate more on reading performance on the best performing OSDs, causing the OSDs to become a “hot spot”. Based on the predictive model and the TOPSIS mathematical model, we get the relative closeness of the OSDs in real-time and then make optimization decisions. This model avoids this “hot spot” problem. In Fig. 7b, horizontal coordinates 2, 3, and 4, the write performance of the cluster remains unchanged.

Figures 6a and 7a show the normalized random read and write throughputs for four different classes of OSDs, respectively. When we measured the performance of each type of OSDs, the cluster selected only two OSDs of the same type to provide read and write services. The read and write performance of the 1 and 2 types of OSDs (corresponding to horizontal coordinates 1 and 2) is almost equal and less than the baseline read and write performance. Type 3 and type 4 OSDs (corresponding to horizontal coordinates 3 and 4) have almost equal read and write performance and are greater than the baseline read and write performance. Type 1 and type 2 OSDs are both HDD-type disks. Type 3 and type 4 OSDs are both SSD-type disks. The main difference between the first two types of OSDs and the last two types of OSDs is the computational power of the node on which they reside. Since the nodes did not reach the upper limit of computing power during our testing, the impact on OSD performance was insignificant.

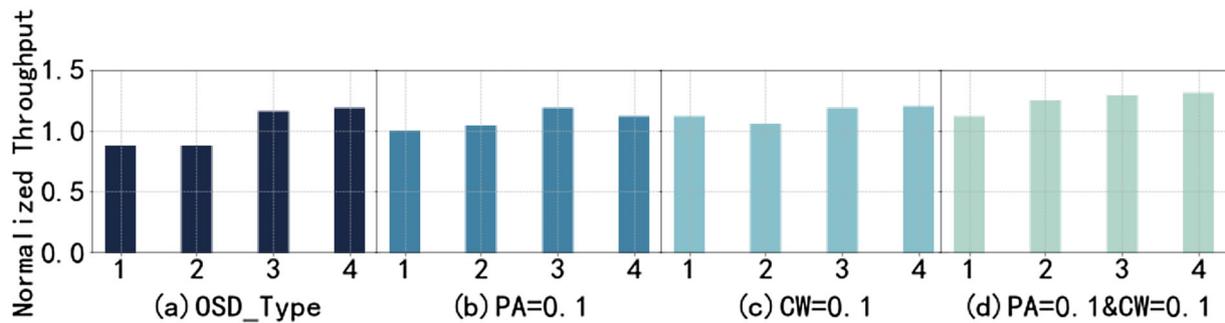
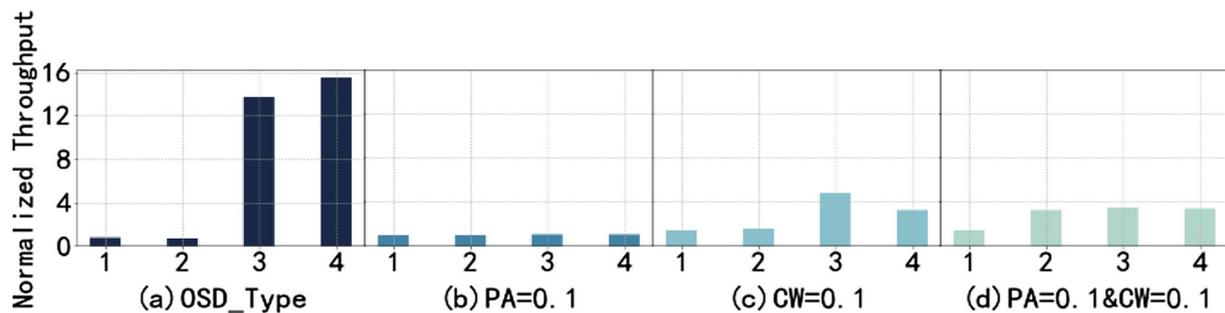
As shown in Fig. 6c and Fig. 7c, at horizontal coordinate 1, we update the CW values of all OSDs to 1 and restrict the range of values to [0,1]. This operation allows us to easily observe the impact of changes in CW values on

Table 1 Physical environment hardware configuration

Ceph node1 (×4)	
Processor	Intel (R) Core (TM) i7-4790 @ 3.60 GHz
Memory	16 GB
Network	1 Gb/s
OS	CentOS Linux release 7.3.1611 (Core)
Ceph node2 (×2)	
Processor	Intel (R) Core (TM) i77700 @ 3.60 GHz
Memory	8 GB
Network	1 Gb/s
OS	CentOS Linux release 7.3.1611 (Core)
SDN Controller (×2)	
Processor	Intel (R) Core (TM) i7-4790 @ 3.60 GHz
Memory	16 GB
Network	1 Gb/s
OS	Ubuntu 20.04.3 LTS
Client node (×2)	
Processor	Intel (R) Core (TM) i7-4790 @ 3.60 GHz
Memory	16 GB
Network	1 Gb/s
OS	CentOS Linux release 7.3.1611 (Core)

Table 2 OSD hetero-geneous resources classification

OSD	bw	cpu	mem	Host	Type	Set β_i	OSD_Type
{4, 6, 12, 17, 18, 23}	1	8	8	3	1	{1, 8, 8, 3, 1}	1
{8, 9}	1	16	8	1	2	{1, 16, 8, 1, 2}	2
{10, 16}	1	16	8	2	1	{1, 16, 8, 2, 1}	3
{7, 30}	1	16	8	2	2	{1, 16, 8, 2, 2}	4

**Fig. 6** Comparison of the random read normalized throughput of the heterogeneous cluster under different constraints**Fig. 7** Comparison of the random write normalized throughput of the heterogeneous cluster under different constraints

performance. The read and the write performance improve by 12% and 41%, respectively. Because SSD-type OSDs typically have less storage capacity than HDD-type OSDs, SSD-type OSDs carry fewer PGs and less read and write load. At this point, the number of PGs taken by OSDs tends to be the same, and the read/write load also tends to be the same. However, SSD-type OSDs are more capable of handling reads and writes. This operation increases the overall reading and writing performance of the cluster. At horizontal coordinate 2, we update the CW value of the first type of OSDs to 0.1. This operation means more PGs are distributed among the other three types of OSDs with better performance to complete read and write requests. The read and the write performance improve by 6% and 57%, respectively. At horizontal coordinate 3, we update the CW value to 0.1 for both type 1 and type 2 OSDs. The read and the write performance improve by 19% and 382%, respectively. In horizontal coordinate 4, we update the CW values of Type 1, Type 2, and Type 3 OSDs to 0.1. The read and the write performance improve by 20% and 222%,

respectively. Overall, as I keep updating the CW values for the different categories of OSDs, the improvement in reading performance keeps increasing while the write performance increases and decreases. This change indicates that there is also a “hot spot” problem.

As shown in Fig. 6d and Fig. 7d, at horizontal coordinate 1 we initialize all OSDs with PA and CW values of 1, both taking values in the range [0,1]. The read and the write performance improve by 12% and 41%, respectively. At horizontal coordinate 2, we update the PA and CW values of the first class of OSDs to 0.1. This operation means we concentrate the read load and migrate more PGs to complete read and write requests on the other three types of better-performing OSDs. Read and write performance is improved by 25% and 225%, respectively. At horizontal coordinate 3, we update the PA and CW values of the first and second-class OSDs to 0.1. Read and write performance improves by 29% and 248%, respectively. In horizontal coordinate 4, we update the PA and CW values of the first three types of OSDs to both 0.1. The read and the write

performance improve by 31% and 343%, respectively. Overall, as I keep updating the PA and CW values for the different categories of OSDs, the read performance improvement keeps increasing, and the writing performance improves and then decreases. This change indicates that there is also a “hot spot” problem. In horizontal coordinates 2, 3, and 4, we get that the overall read performance of the cluster exceeds that of the highest read performance type 3 and type4 OSDs. As the number of OSDs gets more extensive, each OSD shares less of the read load since the read operations only end on the primary OSD.

5.3 Heterogeneous system load monitoring

We divide the data collection into two phases. Phase 1: The SDN controller obtains the information of the storage nodes through active and passive methods. To prevent network congestion caused by monitoring packets sent by the SDN controller, the time interval for monitoring in an active way is 1 s in the experiment. And the time interval for the passive way is equal to the time the storage node spends to collect its information (not a fixed value). Phase 2: Ceph monitor node remotely acquires data temporarily stored on SDN controller. We calculate the time the SDN controller takes to collect information from each storage node by collecting several data sets. Then the time is extended by about 10% as the cycle time of the remote call of the Ceph monitor node. To avoid consecutive calls to the SDN controller by the Ceph monitor node collecting the same data set, we extend this time by a further 10% as the cycle time. During the two stages of data collection, each OpenFlow monitoring packet sent was only 32 bytes. The bandwidth occupied is less than 1/10000 of the total bandwidth, so it is negligible to the network. In addition, OpenFlow switches, SDN controllers, and Ceph monitoring nodes are parallelized while processing their transactions. The size of OpenFlow monitoring packets periodically sent by the SDN controller and Ceph storage node is 32 bytes. The measurement delay generated in sending packets is at the level of microseconds. We could ignore the influence on the millisecond delay.

5.4 Build system adaptive read/write optimization algorithm

In this subsection, we first build the predictive model for OSDs. Then we obtain the predictive performance weights of the multi-attribute decision mathematical model for use in the adaptive read–write optimization algorithm.

5.4.1 Build heterogeneous OSDs performance prediction models

In building the prediction model, set the step size s to 1 and the number of generated vector sets p to 1000. When the selected OSD dataset is collected, the load size of the OSD increases by s each time. If the CW value of the selected OSD is repeatedly increased in steps of s until the cluster IOPS no longer grows, the CW value is N . The parameter s means that the OSD’s load acquisition interval $[1, N]$ is divided into $(N - 1)/s$ parts. A smaller s indicates a higher density of load data set acquisition. The parameter p denotes the number of load data sets collected under the OSD’s particular read and write load. The size of the parameter p takes the inflection point where the prediction model accuracy rises and falls or rises and stays the same. This experiment takes $p \in [1, 1500]$, and multiple investigations take the average of the prediction model accuracy to get a better result with $p = 1000$. For each OSD in the set χ , we use the passive way of the SDN controller to upload the storage node load information and repeat the generation of the corresponding parameter sets $\{\text{consume}_i, r_{\text{io}_i}, \text{IOPS}_i\}$ and $\{\text{consume}_i, w_{\text{io}_i}, \text{IOPS}_i\}$, $1 \leq i \leq 1000$. The value of Crush Weight is repeatedly increased by s until the cluster’s overall read/write performance does not increase or reach the predetermined performance requirement. Finally, we construct the OSD’s four-vector training sets S_6, S_9, S_{10} , and S_{30} . Table 3 shows the OSD vector set S part of the collected information. The adaptive algorithm can directly obtain the optimization decision results in different application scenarios if we receive sufficient data sets to establish the prediction model. Suppose the prediction model needs dynamic training, such as in this experiment. In that case, the training time of each type of OSD is about 1 s. The interval between decisions of the optimization algorithm is half an hour. This decision time shows that the prediction model evaluation time is much less than the algorithm decision time.

In collecting OSD load information, we use an FIO write and read client to read and write an RBD block simultaneously. We use the dictionary IOPS $\{r_{\text{iops}}, w_{\text{iops}}\}$ to record the read IOPS and write IOPS. It is important to note that more clients in a cluster read and write to an RBD block simultaneously. The more likely it is that a single OSD multiplexing will lead to overload. This paper proposes a performance prediction model with the OSD resource consumption set. It uses the OSD resource residual set to make adaptive optimization decisions. This approach has two benefits: (1) we use the OSD resource residual set to make optimization decisions in favor of cluster load balancing. (2) we normalize the read and write operations of multiple clients to the same OSD

into the total number of r_io and w_io . Then we build a performance prediction model for the OSD with better adaptability. The model shields the differences caused by different numbers of client read/write operations.

5.4.2 Build multi-attribute decision models

We construct a performance prediction model using the random forest algorithm to optimize the Ceph system automatically. In Sect. 4.5.1, the parameters k , m , and d denote the maximum number of features of the decision tree parameters, the number of decision trees, and the depth of the decision trees, respectively. A higher value of m leads to higher accuracy of the random forest model and a longer model evaluation time [34], setting $m \in [1, 200]$. Larger values of d can lead to overfitting of the model set $d \in [1, 10]$. Set $k \in [1, 10]$ and import grid search cross-validation. Network search allows the model parameters to be traversed according to our given list to find the model that works best, and the cross-validation tells us the accuracy of the model. To reflect the model accuracy more intuitively, we define the prediction accuracy formula [35] as:

$$precision = \left(1 - \frac{\sum_{i=1}^n (\widehat{pre_io}_i - io_i)^2}{\sum_{i=1}^n (io_i - \bar{io})^2} \right) \times 100\% \quad (12)$$

The parameter $\widehat{pre_io}_i$ is the performance r_io or w_io of the predicted OSD. The parameter \bar{io} is the mean value of the OSD's true performance r_io or w_io . The parameter io_i is the true performance r_io or w_io of the OSD. The above Eq. (12) numerator indicates the residual of the performance predicted by the predicted value. The denominator indicates the residual of the performance obtained by predicting all data with the sample mean.

Table 3 OSD vector set S part acquisition information

OSD	bw	cpu	mem	pgs	r_io	w_io	IOPS
6	29.82	0.2	1.03	75	0	120	{1862, 53}
...							
6	21.14	0.13	2.18	124	12	340	{762, 38}
9	16.4	0.06	0.98	80	8	60	{1862, 53}
...							
9	8.48	0.03	2.26	382	272	788	{968, 67}
10	56.31	0.13	2.7	290	4	600	{1862, 53}
...							
10	71.6	0.14	2.78	371	112	1656	{3201, 250}
30	65.55	0.1	1.42	374	148	2060	{1862, 53}
...							
30	7.24	0.03	0.66	98	4	348	{3242, 264}

When the precision < 0 , the residual of the result predicted by the model is larger than the residual obtained by the benchmark model (predicting all data with the sample mean). The result means that the model predicts the result very poorly. When the precision > 0 , the larger the precision, the smaller the numerator. The result indicates that the residuals of the performance prediction results are smaller, and the performance prediction effect is better. When precision = 1, the result indicates that the model prediction data completely fits the actual data as the best ideal fit model.

According to Fig. 5, we build the performance prediction models of the four types of OSDs in the Ceph cluster by training the vector sets S6, S9, S10, and S30 of the four OSDs, respectively. Finally, the values of accuracy precision of r_io and w_io corresponding to the established OSD performance prediction models are shown in Table 4, respectively.

When the model is as accurate as expected, we build the prediction model `rf_reg`. Under varying load states,

we can obtain the performance of r_io and w_io predicted by OSD. We then used the `rf_reg` model to analyze the metric performance importance. Tables 5 and 6 show that we obtain the OSD predicted read and write performance weights corresponding to the load factors affecting OSD.6, OSD.9, OSD.10, and OSD.30, respectively.

As can be seen from Table 5, the HDD-type OSD consumes 2 to 9 times more bandwidth weight than the SSD-type OSD consumes when reading data objects. The CPU weight has the same consumption trend. However, the SSD-type OSD has 0.5 to 4 times more weight in the number of PGs than the HDD-type OSD. The mem weight has the same consumption trend.

As seen in Table 6, the HDD-type OSDs consume slightly less bandwidth weight and mem weight than SSD-type OSDs when writing data objects. There is no clear pattern in the weight of CPU resources and the weight of the number of PGs owned.

5.5 Performance evaluation

Figures 8a and b, 9a and b represent the normalized throughput curves and latency curves obtained experimentally for simultaneous 100% random write operations and 100% random read operations on the same RBD block at different workloads. In the designed prototype system, TOPSIS_PA, TOPSIS_CW, and TOPSIS_PACW algorithms improve the throughput of reading operations by 19 to 40%, 23 to 60%, and 36 to 85%, and reduce latency by 16 to 29%, 19 to 37% and 26 to 46%, respectively. The TOPSIS_PA, TOPSIS_CW and TOPSIS_PACW algorithms improve write throughput by 2 to 4%, 180 to 468% and 188 to 611%, and reduce latency by 2 to 4%, 64 to 82%

and 65 to 86%, respectively. Overall the TOPSIS_PACW algorithm provides better elastic reading and writing performance.

Compared with the original Ceph system, the TOPSIS_PA algorithm improves read performance by 36% and reduces latency by 29% while ensuring the data distribution balance and writing load of the Ceph system remain unchanged.

The improved CRUSH algorithm completes the distribution of PG on the OSD in the first stage, still using storage capacity as the CW value. In contrast, the second stage introduces the OSD's remaining network bandwidth and load state as constraints to obtain the PA value. And the PA value determines the result of selecting the primary OSD when the system performs a read operation. According to the load status of the OSD monitored by the SDN in real time, the system adjusts the OSD with low load or good performance as the primary OSD to complete the data transfer. Hence, it has higher throughput and lower latency.

However, there is little improvement in writing operations. The Ceph's write operations include collaborative completion among multiple components, such as messenger layer communication, internal PG processing in the OSD, metadata logging, and synchronization operations for writing data objects to file storage. In addition, the completion of the write operation for multiple replicas is marked by the end of the multiple OSDs with the highest write latency.

Therefore, even if the good-performing OSDs concentrate on more read operations, the read load of the poor-performing OSDs is reduced. This transfer of reading load does not increase the throughput of the write operations as they include several components. In practice, the TOPSIS_PA algorithm is suitable for high read performance requirements and sensitive data distribution balance applications.

TOPSIS_CW algorithm mainly provides the elastic write performance of the system. According to the requirements of application scenarios, it can adaptively sacrifice the uniform distribution of some data in exchange for efficient elastic read and write performance. It improves

Table 4 OSD performance prediction model accuracy

Vector Set: {S}	Precision(r_io)	Precision(w_io)
{S ₆ }	96.31	99.34
{S ₉ }	96.10	93.55
{S ₁₀ }	94.93	92.19
{S ₃₀ }	95.68	96.20

Table 5 OSD predictive read performance weights

Weight set	bw	cpu	mem	pgs	w_io
{R ₆ }	0.42	0.17	0.12	0.18	0.11
{R ₉ }	0.63	0.16	0.05	0.10	0.06
{R ₁₀ }	0.20	0.08	0.15	0.26	0.31
{R ₃₀ }	0.07	0.02	0.19	0.44	0.28

Table 6 OSD predictive write performance weights

Weight Set	bw	cpu	mem	pgs	r_io
{W ₆ }	0.21	0.08	0.22	0.31	0.18
{W ₉ }	0.11	0.02	0.19	0.57	0.12
{W ₁₀ }	0.28	0.01	0.14	0.37	0.20
{W ₃₀ }	0.25	0.07	0.18	0.39	0.11

the elastic write performance by 180 to 468% and reduces the latency by 64 to 82%. And it improves the read performance by 23 to 60% and reduces the latency by 19 to 37%. The improved CRUSH algorithm introduces the OSD's remaining network bandwidth and load state as constraints in PGs' first stage of OSD selection. The PGs are redistributed on the OSDs using the CW values of multi-attribute decisions as weights to complete the distribution. At this time, more PGs redistribute to the high-performance OSDs. The TOPSIS_CW algorithm greatly reduces the coworking time for write operations among multiple components, resulting in higher write throughput and lower latency for the storage system. Meanwhile, the result of Ceph read operation to select primary OSD is determined by the PA weight value in the second stage. However, the CW value redistributes the first-stage PGs on the OSD. Hence, the storage system gets higher read throughput and lower latency. In practice, the TOPSIS_CW algorithm is suitable for applications that require high write performance but are not sensitive to data distribution imbalance.

The TOPSIS_PACW algorithm considers the optimization ideas of the TOPSIS_PA and the TOPSIS_CW. It improves elastic read performance by 36 to 85% and reduces latency by 26 to 46%. And it improves elastic write performance by 188 to 611% and reduces latency by 65 to 86%. It is important to note that in the case of node heterogeneity, the performance PA value is essentially the position of dynamically adjusting the selection of reading primary OSD. Ceph adaptively places them on OSDs with excellent read performance or low load. In comparison, the performance CW value is essentially the result of

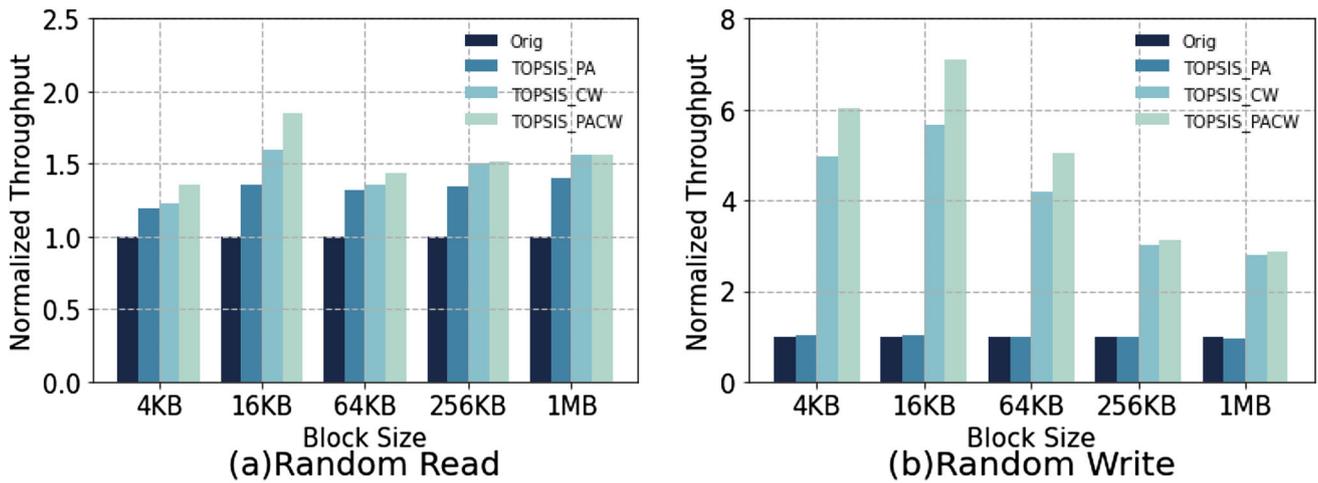


Fig. 8 Comparison of the normalized throughput of the TOPSIS series algorithm at different workloads

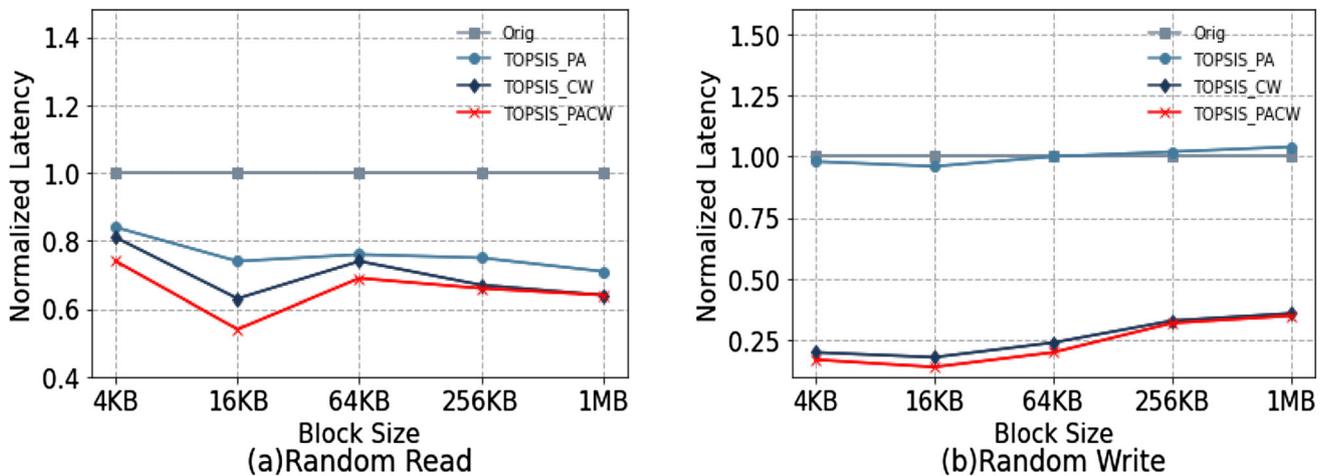


Fig. 9 Comparison of the normalized latency of the TOPSIS series algorithm at different workloads

dynamically adjusting the choice of OSD combination when writing all replica data objects. The CW’s change could adaptively migrate the PGs carrying data objects onto the OSD with superior write performance or low load. As a result, we provide the elastic read and write performance of the Ceph system to meet more Quality of service requirements. It not only ensures reliability but also increases the high availability of the storage system. In practice, the TOPSIS_PACW algorithm is suitable for applications that require high read and write performance but are not sensitive to unbalanced data distribution.

5.6 Advantages and disadvantages of the TOPSIS series algorithms

In the designed prototype system, the OSDs in the Ceph system have varying upper limits on reading and writing performance. When optimizing read and write performance

using Load Balancing Strategy, this paper proposes the TOPSIS series of algorithms based on performance prediction and multi-attribute decision making. It can guarantee the load balancing of {Pool₁, Pool₂, ..., Pool_n} of the same type of OSDs and the load balancing of different types of OSDs in Dynamic Pool separately. When we use OSDs with good performance to optimize read and write performance, we should avoid the “hot spot” problem. For example, the TOPSIS_PACW keeps the same performance as TOPSIS_CW when optimizing the read and write performance of 256 k and 1 M objects. This trend in performance is a “hot spot” on the best-performing OSDs.

Therefore, we need to use the load state prediction model for different classes of OSDs to make a predictive evaluation of the effect before each optimization. If predictive evaluation contradicts the expected optimization goal, Algorithm 4 will revoke the operation. This kind of performance prediction model trained based on the actual

load state of OSDs has good reference and adaptability to make performance optimization adjustments more reasonable. In addition, we also consider the impact of data migration and recovery in the cluster. Then we can make corresponding adjustments to ensure that the OSD prioritizes the execution of data migration and recovery operations or the execution of optimized read and write performance operations. So the system read and write performance has a more fine-grained granularity.

Furthermore, Dynamic Pool adaptively migrates PGs to $Pool_1 \sim Pool_n$ storage pools by on-demand. The on-demand means that if the range of reading and writing IOPS that $Pool_i$ can provide is $[a, b]$, we place the appropriate customer demand in $Pool_i$ to reduce wasted resources. Finally, we need to discuss how to use the relationship between data distribution and nodes' read/write performance in the ceph system. To ensure system reliability, we adaptively adjust the system read and write performance to make system availability more robust.

6 Conclusion

For Ceph heterogeneous storage systems, we design a network-aware approach based on software-defined networking technology. To improve the CRUSH Algorithm, we propose an adaptive read/write optimization algorithm for Ceph heterogeneous systems via performance prediction and multi-attribute decision-making. Firstly, the OSDs are classified based on the Node Heterogeneous Resource Partitioning Strategy. Then the prediction model is established by combining the load state of OSD. Finally, we complete the selection of OSDs by solving the mathematical model of multi-attribute decision-making. We also propose a Dynamic Pool to meet clients' dynamic read and write performance needs. The experimental results show that TOPSIS_PA improves read performance by 36%. TOPSIS_CW and TOPSIS_PACW algorithms improve the elastic write performance by 180 to 468% and the elastic read performance by 23 to 60%. In summary, the TOPSIS series of algorithms ensures system reliability while adding high availability.

To meet the requirements of a large-scale storage network, we have implemented the cooperative communication deployment of multiple controllers in an SDN network. It can solve the performance bottleneck problem of a single SDN controller and improve the reliability of the cluster. Furthermore, We need to study the overhead of the SDN monitoring process (switch channel bandwidth consumption, etc.) and evaluate the establishment time of prediction models of different scales. At the same time, the network state changes dynamically, so it is necessary to consider the network state measurement and decisive

moment in fine granularity to optimize the cluster performance. We need to dynamically and rationally determine the different load factor dimension weights. These are more meaningful research tasks that need to carry out subsequently.

7 Intellectual property

We confirm that we have given due consideration to the protection of intellectual property associated with this work. There are no impediments to publication concerning intellectual property. In so doing, we confirm that we have followed the regulations of our institutions concerning intellectual property.

Author contributions The main author ZL contributed to the design and implementation of the research, the analysis of the results, and the writing of the manuscript. YW supervised and analyzed the experimental results. And he also reviewed the manuscript.

Funding This work is supported by the National Natural Science Foundation of China (6166 2018, 6166 1015, 61831013), the China Postdoctoral Science Fund Project (2016M602922 XB), Guangxi Innovation-driven Development Special Project (Science and Technology Major Special Project Gui Ke AA18118031), Guilin University of Technology Research Start-up Fund Project (GUTQDJJ2017-2000019), Innovation Project of Guangxi Graduate Education(YCSW2021179).

Data availability The data used to support the findings of this study is available from the corresponding author upon request.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Heidari, A., et al.: Internet of things offloading: ongoing issues, opportunities, and future challenges. *Int. J. Commun. Syst.* **33**(14), e4474 (2020)

2. Akter, S., Wamba, S.F.: Big data analytics in E-commerce: a systematic review and agenda for future research. *Electron. Mark.* **26**(2), 173–194 (2016)
3. Heidari, A., Navimipour, N.J.: Service discovery mechanisms in cloud computing: a comprehensive and systematic literature review. *Kybernetes* **51**, 952–981 (2021)
4. Heidari, A., Navimipour, N.J.: A new SLA-aware method for discovering the cloud services using an improved nature-inspired optimization algorithm. *PeerJ Comput. Sci.* **7**, e539 (2021)
5. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system[C]//Proceedings of the nineteenth ACM symposium on Operating systems principles, 29–43 (2003)
6. Weil, S.A., Brandt, S.A., Miller, E.L., et al.: Ceph: a scalable, high-performance distributed file system [C]//Proceedings of the 7th symposium on Operating systems design and implementation, 307–320 (2006)
7. Huang, C., Simitci, H., Xu, Y., et al.: Erasure coding in windows azure storage[C]//2012 USENIX Annual Technical Conference (USENIX ATC 12), 15–26 (2012)
8. Palankar, M.R., Iamnitchi, A., Ripeanu, M., et al.: Amazon S3 for science grids: a viable solution? [C]//Proceedings of the 2008 international workshop on Data-aware distributed computing, 55–64 (2008)
9. Bollig, E.F., Allan, G.T., Lynch, B.J., et al.: Leveraging openstack and ceph for a controlled-access data cloud[M]//Proceedings of the practice and experience on advanced research computing, 1–7 (2018)
10. Weil, S.A., Brandt, S.A., Miller, E.L., et al.: CRUSH: controlled, scalable, decentralized placement of replicated data[C]//SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. IEEE, 31–31 (2006)
11. Chum, S., Park, H., Choi, J.: Supporting SLA via adaptive mapping and heterogeneous storage devices in Ceph. *Electronics* **10**(7), 847 (2021)
12. Karger, D., Lehman, E., Leighton, T., et al.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web[C]//Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 654–663 (1997)
13. Chen, T., Xiao, N., Liu, F.: An efficient hierarchical object placement algorithm for object storage systems. *J. Comput. Res. Dev.* **49**(4), 887 (2012)
14. Jia, C.J., Wang, Y., Mendl, C.B., et al.: Paradeisos: a perfect hashing algorithm for many-body eigenvalue problems. *Comput. Phys. Commun.* **224**, 81–89 (2018)
15. Jeong, B., Khan, A., Park, S.: Async-LCAM: a lock contention aware messenger for Ceph distributed storage system. *Clust. Comput.* **22**(2), 373–384 (2019)
16. Qian, L., Tang, B., Ye, B., et al.: Stabilizing and boosting I/O performance for file systems with journaling on NVMe SSD. *Sci. China Inf. Sci.* **65**(3), 1–15 (2022)
17. Yang, C.T., Chen, S.T., Cheng, W.H., et al.: A heterogeneous cloud storage platform with uniform data distribution by software-defined storage technologies. *IEEE Access* **7**, 147672–147682 (2019)
18. Kong, L.W., Moreno, O.: Characterization and prediction of performance loss and MTTR during fault recovery on scale-out storage using DOE & RSM: a case study with Ceph. *IEEE Trans. Cloud Comput.* **9**(2), 492–503 (2018)
19. Zhang, Y., Debroy, S., Calyam, P.: Network measurement recommendations for performance bottleneck correlation analysis[C]//2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN). IEEE, 1–7 (2016)
20. Clegg, R.G., Withall, M.S., Moore, A.W., et al.: Challenges in the capture and dissemination of measurements from high-speed networks. *IET Commun.* **3**(6), 957–966 (2009)
21. Tootoonchian, A., Ghobadi, M., Ganjali, Y.: OpenTM: traffic matrix estimator for OpenFlow networks [C]//International Conference on Passive and Active Network Measurement, pp. 201–210. Springer, Berlin, Heidelberg (2010)
22. Liberato, A., Martinello, M., Gomes, R.L., et al.: RDNA: residue-defined networking architecture enabling ultra-reliable low-latency datacenters. *IEEE Trans. Netw. Serv. Manage.* **15**(4), 1473–1487 (2018)
23. Kafetzis, D., Vassilaras, S., Vardoulas, G., et al.: Software-defined networking meets software-defined radio in mobile Ad hoc networks: state of the art and future directions. *IEEE Access* **10**, 9989–10014 (2022)
24. Girisanekar, S.T., Truong-Huu, T., Gurusamy, M.: SDN-based dynamic flow scheduling in optical data centers[C]//2017 9th International Conference on Communication Systems and Networks (COMSNETS). IEEE, 190–197 (2017)
25. Weil, S.A., Leung, A.W., Brandt, S.A., et al.: Rados: a scalable, reliable storage service for petabyte-scale storage clusters[C]//Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07, 35–44 (2007)
26. Honicky, R.J., Miller, E.L.: Replication under scalable hashing: a family of algorithms for scalable decentralized data distribution[C]//18th International Parallel and Distributed Processing Symposium, 2004. Proceedings. IEEE, 96 (2004)
27. Liu, G., Liu, X.: The Complexity of Weak Consistency[C]//International Workshop on Frontiers in Algorithmics, pp. 224–237. Springer, Cham (2018)
28. Yong, W., Miao, Ye., Qian, He., WenJie, K.: Based on software-defined networking and multi-attribute decision-making node selection method for Ceph storage systems. *J. Comput. Sci.* **42**(434(02)), 93–108 (2019)
29. Wu, L., Zhuge, Q., Sha, E.H.M., et al.: BOSS: An efficient data distribution strategy for object storage systems with hybrid devices. *IEEE Access* **5**, 23979–23993 (2017)
30. Watkins, L.A.: Using network traffic to infer CPU and memory utilization for cluster grid computing applications. (2010). <https://doi.org/10.57709/1347999>
31. Bei, Z., Yu, Z., Zhang, H., et al.: RFHOC: a random-forest approach to auto-tuning Hadoop's configuration. *IEEE Trans. Parallel Distrib. Syst.* **27**(5), 1470–1483 (2015)
32. Chen, Yu., Ying-Chi, M.: Based on random forests and genetic algorithms, automatic tuning of Ceph parameters. *Comput. Appl.* **40**(2), 347–351 (2020)
33. Chen, S.J., Hwang, C.L.: Fuzzy multiple attribute decision making methods. In: *Fuzzy Multiple Attribute Decision Making*, pp. 289–486. Springer, Berlin, Heidelberg (1992)
34. Bei, Z., Yu, Z., Luo, N., et al.: Configuring in-memory cluster computing using random forest. *Future Gener. Comput. Syst.* **79**, 1–15 (2018)
35. Menard, S.: Coefficients of determination for multiple logistic regression analysis. *Am. Stat.* **54**(1), 17–24 (2000)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Zhike Li received his B.S. degree from the Henan University of Technology. He is currently pursuing his M.S. degree at Guilin University of Electronic Science and Technology. His main research interests are cloud computing and distributed storage.



Yong Wang received his Ph.D. degree from East China University of Science and Technology in 2005. He is a full professor with the School of Computer Science and Information Security, Guilin University of Electronic Technology. His research interests include cloud computing, distributed storage system and information security.